Exhibit JX22

TON Development Status
Overall progress towards the test version: 90%

January 28, 2019

1. TON Virtual Machine (TVM)

TON VM or TVM is the component required for executing smart contracts in the TON Blockchain.

✓ *Implementation:* **95% complete**

TVM has been fully implemented and internally tested. Minor modifications will likely be necessary during the process of binding TVM with the TON Blockchain block generation and validation software.

In addition to TVM itself, a database required for storing on disk and accessing large amounts of TVM data (e.g., smart-contract code and data, old blocks, blockchain state) without loading all of it into memory has been developed.

✓ *Documentation:* **95% complete**

The current version of TVM is fully described in *Telegram Open Network Virtual Machine (September 5, 2018)*. Minor modifications to the implementation may require corresponding changes in the documentation.

2. TON Network

The TON Network is the component required for delivering requests (e.g., proposed transactions) and propagating newly-generated TON Blockchain blocks through the network.

✓ *ADNL (low-level overlay network protocol running over IP networks):* **80% complete**

All functionality required for the test version is complete, including elliptic curve cryptography and the node lookup protocol. Some sophisticated options and additional cryptographic options that are not required for the launch of the test version will be implemented later (prior to the final launch).

✓ *Overlay networks over ADNL:* **100% complete**

Overlay networks are required to build node groups inside the ADNL networks. For instance, the validators for a shardchain create their separate overlay network to propagate new block candidates and run a TON-specific Byzantine Fault Tolerant (BFT) consensus protocol.

✓ *Broadcast protocols for overlay networks over ADNL:* **100% complete**

Simple broadcast protocols are used inside overlay networks to propagate small messages, such as the BFT consensus protocol messages, to all members of an overlay network. These protocols are required for the implementation of validator BFT consensus.

✓ *CATCHAIN protocol:* **100% complete**

The CATCHAIN protocol is a variant of broadcast protocols tailored for implementing BFT consensus protocols and for solving similar group consensus tasks in a closed membership overlay network. As such, it is the first step in implementing the validator BFT consensus protocol.

✓ *Streaming broadcast protocols:* **100% complete**

Streaming broadcast protocols are used to quickly propagate large amounts of data, such as newly-generated TON Blockchain blocks (to all full nodes) and block candidates (to the validators of the corresponding shardchain). Streaming broadcast protocols employed by TON use Forward Error Correction (FEC) protocols as their component.

3. TON Blockchain Block Generation and Validation

The block generation and validation software relies heavily on TVM and the TON Network to create new block candidates, validate them among the validators, and propagate the signed blocks to all full nodes. Since the work on the TVM and TON Network components listed above is largely complete, the TON Blockchain is now in active development.

✓ *Documentation:* **90% complete**

The documentation is intended to present a complete description of the masterchain and shardchain block format in the TON Blockchain. The shardchain block description available in *Telegram Open Network Blockchain (September 5, 2018)*, while unchanged in the principal points, will require some minor modifications based on changes that have arisen during the final development phases. Some details of the masterchain blocks, such as the list of all configurable parameters with their respective types, are not fully documented at this time, because they are not completely finalized yet; they will be added to the documentation during the testing phase.

✓ *Block manipulation library:* **95% complete**

The block manipulation library is intended to store entire blocks and their parts in files, load these data into memory, and access or modify different data structures present in a block. All methods originally intended have been implemented. Some minor modifications may be required during the final stages of validator software development.

✓ *Validator BFT Consensus protocol:* **95% complete**

The TON-specific Byzantine Fault Tolerant (BFT) consensus protocol is used by validators during block generation to reach agreement on the next block of a shardchain or the masterchain (as applicable). This custom BFT protocol, built upon TON CATCHAIN, has been completely implemented and tested, yielding 2–3 second consensus time for a test network of 100 servers distributed around the world. This is consistent with the five second block generation interval proposed in the TON Whitepaper.

▷ *Validator software:* **60% complete**

Validator software uses the block manipulation library to generate block candidates, validate block candidates proposed by other validators, and achieve consensus on the next block in a shardchain or the masterchain. It consists of the network component (especially the BFT Consensus protocol) and of the local block generation (collation) and validation component. The network component is almost complete. The local block generation and validation component is currently halfway complete, but most development efforts are currently dedicated to the completion of this component.

✓ *Full node software:* **80% complete**

A *full node* of the TON Blockchain is a program that obtains and stores local copies of all or some blocks, and may re-distribute these blocks to other full nodes if required. It is also an important component of the validator software, because validators are (specialized) full nodes as well. The network component and the local storage component of the full node software are currently in active development and nearing their completion.

▷ *Smart-contract development, test, and debug environment:* **50% complete**

A test and debug environment for smart contracts is already implemented and internally tested, along with low-level "TVM assembly" smart-contract language. The compiler from a high-level smart-contract language is 20% complete—its core functionality is ready, but more built-in functions and operations need to be defined.

▷ *Fundamental and sample smart contracts:* **20% complete**

Some sample smart contracts are prepared in "TVM assembly". The implementation of fundamental smart contracts—which reside in the masterchain and run crucial tasks such as electing new validators and changing configurable parameters—requires the availability of the high-level smart-contract compilers and development tools discussed above. However, the launch of a test network with very basic versions of fundamental smart contracts is our top priority. We plan to replace the basic versions of these smart contracts with more sophisticated versions during the testing phase.

*** This communication contains forward-looking statements, including statements of plans, objectives, expectations, development status and intentions. Any number of factors could cause actual results to differ materially from those contemplated by any forward-looking statements, including but not limited to the risks identified in Appendix B to the Whitepaper ***